SAMEER MOTWANI
ASSIGNMENT 1
CS-6240-01

Weather Data Results:

For each of the versions of your sequential and multithreaded program detailed in B and C, report the minimum, average, and maximum running time observed over the 10 runs. (5 points)

B1: SEQUENTIAL

Minimum Time :2493.0
Average Time :2731.9
Maximum Time :3071.0

B2: NO LOCK

Minimum Time :1174.0
Average Time :1263.1
Maximum Time :1311.0

B3: COARSE-LOCK

Minimum Time :1220.0
Average Time :1360.8
Maximum Time :1432.0

B4: FINE-LOCK

Minimum Time :1156.0
Average Time :1350.5
Maximum Time :1401.0

B5: NO SHARING

Minimum Time :1202.0
Average Time :1301.0
Maximum Time :1370.0

C1: Sequential with Fibonacci

Minimum Time :10967.0
Average Time :11237.0
Maximum Time :11888.0

C2: NO LOCK with Fibonacci

Minimum Time :5244.0
Average Time :5469.9
Maximum Time :5602.0

C3: COARSE LOCK with Fibonacci

Minimum Time :12980.0
Average Time :13379.5
Maximum Time :14137.0

C4: FINE-LOCK with Fibonacci

Minimum Time :5560.0
Average Time :6152.5
Maximum Time :7730.0

C5: NO SHARING with Fibonacci

Minimum Time :5063.0
Average Time :5834.3
Maximum Time :6642.0

Report the number of worker threads used and the speedup of the multithreaded versions based on the corresponding average running times. (5 points)

The number of worker threads created are 4.
The calculated speed up of the multithreaded versions are stated below:
NO-LOCK: 2.16
NO-LOCK with Fibonacci: 2.05

COARSE-LOCK: 2.00
COARSE -LOCK with Fibonacci: 0.83

FINE-LOCK: 2.022
FINE-LOCK with Fibonacci: 1.82

NO-SHARING: 2.099
NO-SHARING with Fibonacci: 1.92

Answer the following questions in a brief and concise manner: (4 points each)

Que 1: Which program version (SEQ, NO-LOCK, COARSE-LOCK, FINE-LOCK, NO-SHARING) would you normally expect to finish fastest and why? Do the experiments confirm your expectation? If not, try to explain the reasons.

Ans 1: **NO-LOCK** will run the fastest

- As there are no locks and we can utilize maximum parallelism as no worker will have to wait for resource. All workers will update the resource at the same time. hence there is a chance for final average values to be inconsistent.
- The average values confirm my expectations.

Que 2: Which program version (SEQ, NO-LOCK, COARSE-LOCK, FINE-LOCK, NO-SHARING) would you normally expect to finish slowest and why? Do the experiments confirm your expectation? If not, try to explain the reasons.

Ans 2: **SEQ** is expected to be the slowest as each step is executed one after another unlike the parallel approach where the program makes use of the 4 cores thus making it faster. The average values confirm my expectations.

Ques 3: Compare the temperature averages returned by each program version. Report if any of them is incorrect or if any of the programs crashed because of concurrent accesses.

Ans 3: The average values returned for **NO-LOCK** mechanism are incorrect because:

- Program crashes sometimes because there is lack of synchronization between threads which leads to one thread trying to read a value in data structure which is not present.
- Since there is no lock involved, multiple threads may update a data structure based on old record leading to overwriting of one record over other this leads to inconsistency or wrong results.

Ques 4: Compare the running times of SEQ and COARSE-LOCK. Try to explain why one is slower than the other. (Make sure to consider the results of both B and C—this might support or refute a possible hypothesis.)

Ans 4: **SEQ** is generally slower than **COARSE-LOCK**. But, we have seen that when delay fibonacci (17) is introduced, sequential program runs faster because:

- When input is small, the tasks are performed quickly with less transfer of locks between the threads. Hence parallelism is used to speed up the computation.
- But, as the inputs increase, the wait time for threads increases as they have to wait more often for other thread to release locks. This defeats the improvement in performance we obtain by running a program parallel.

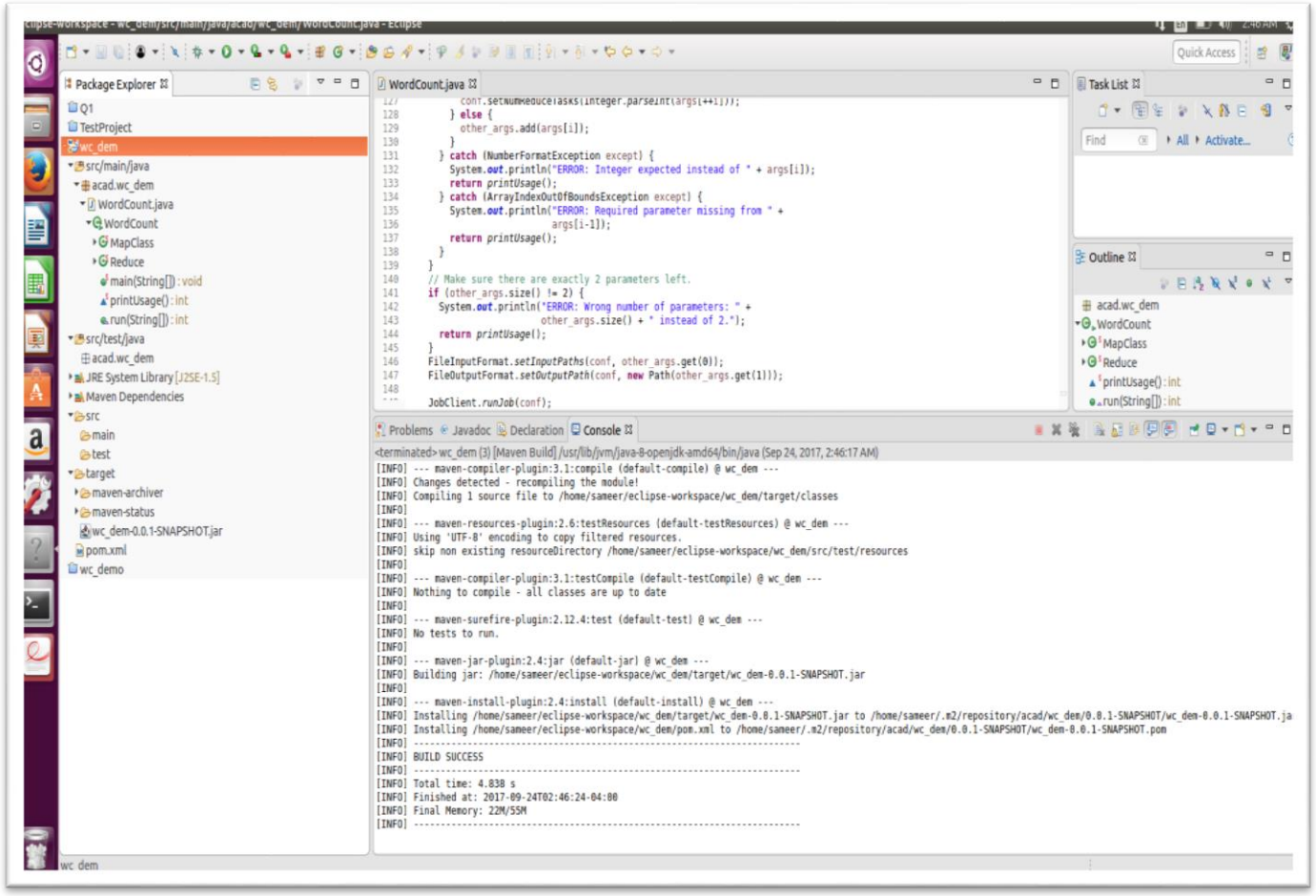
Ques 5: How does the higher computation cost in part C (additional Fibonacci computation) affect the difference between COARSE-LOCK and FINE-LOCK? Try to explain the reason.

Ans 5: **In COARSE-LOCK**, the lock is held over the entire map and this makes sure that no other thread is trying to update the map. Therefore, it can be assured that no two Fibonacci computations will happen in parallel. Hence, all delays will happen in sequential way. However, for **FINE-LOCK** there is a chance

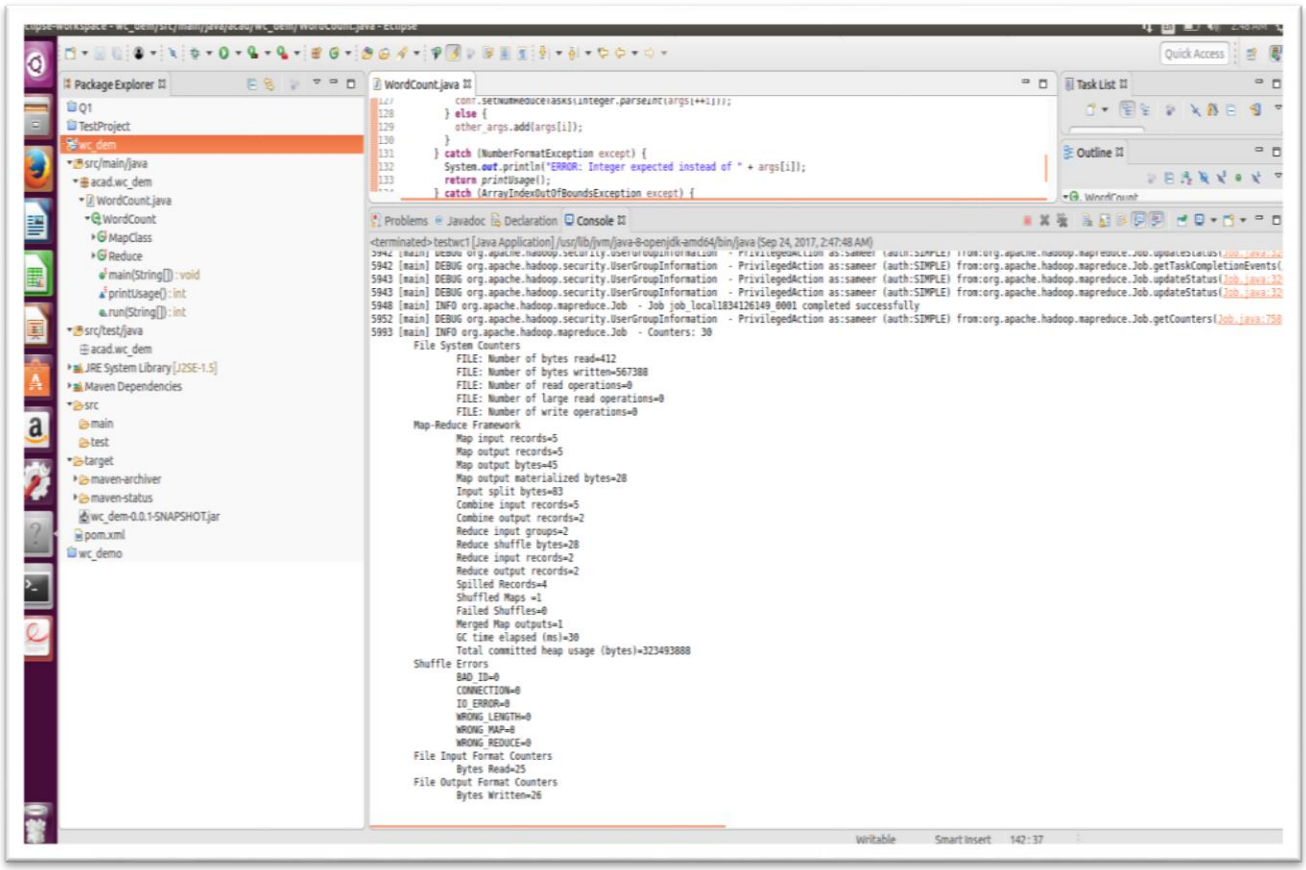
that some Fibonacci computations may happen in parallel since each thread holds lock over the only the variable which it updates and the complete map. Hence with higher computation cost, the **FINE-LOCK** is faster than **COARSE-LOCK**.

WORD COUNT LOCAL:

Screenshot showing directory structure

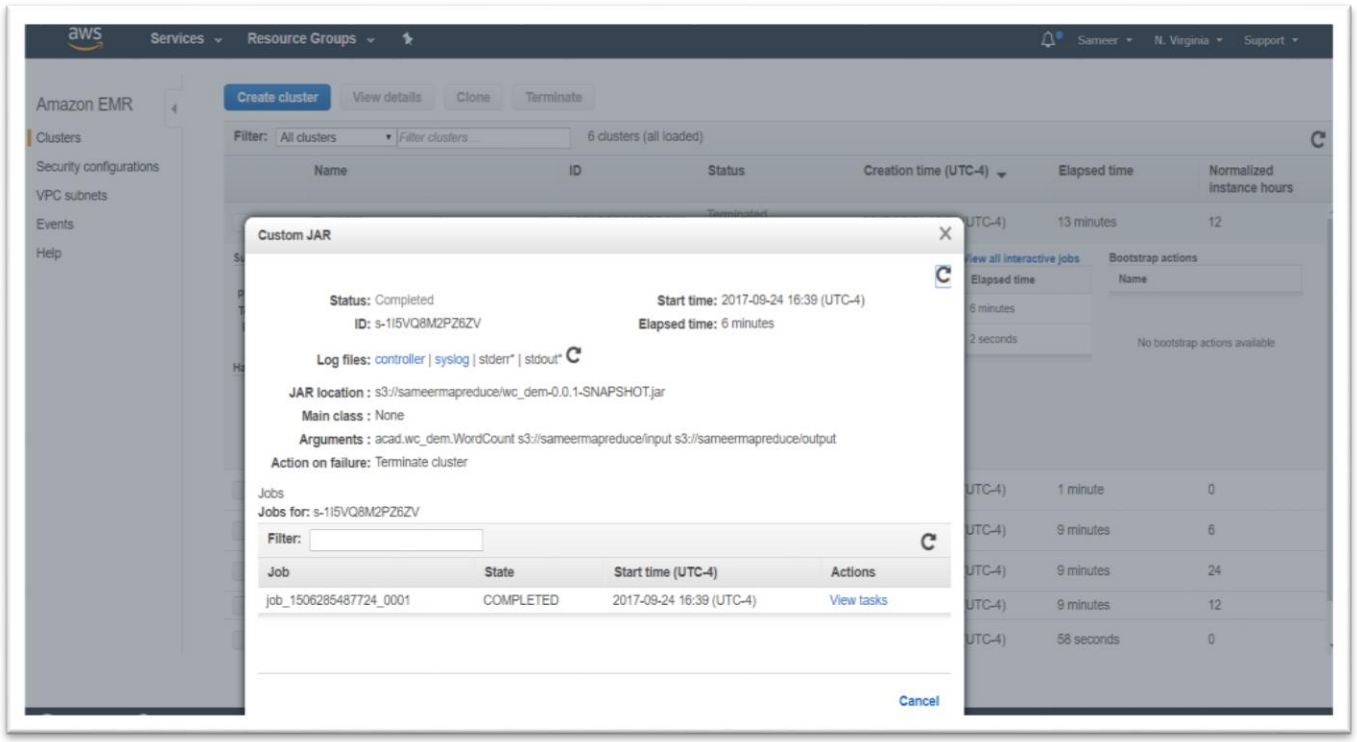


Screenshot of last 20 lines of execution in Eclipse console

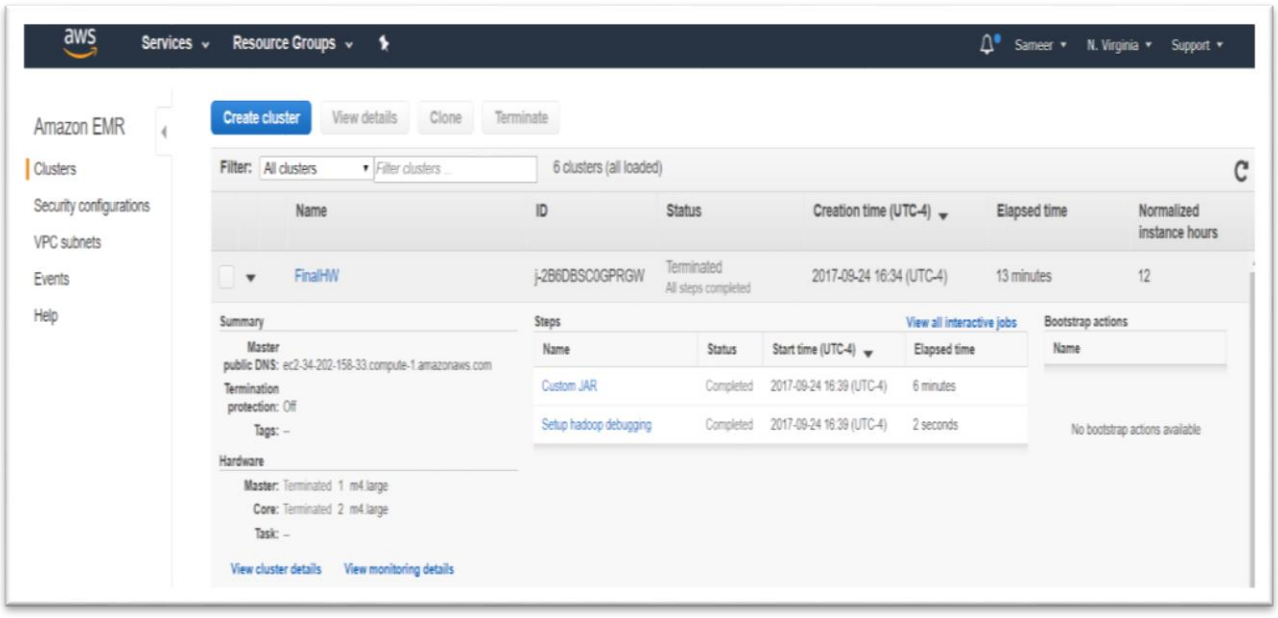


WORDCOUNT AWS

Screenshot of successful execution on AWS



Screenshot of Cluster details



The controller.txt and syslog.txt files are present in the logs folder. The final results produced on AWS are present in results folder.